HES

VIC FORTH™

By Tom Zimmer

FORTH is an interactive language that is many times faster than BASIC, yet is easier to use than assembly language. FORTH programs are modular permitting structured programming and are extremely efficient in memory usage.

VIC FORTH is a complete implementation of the popular Fig version of FORTH.

RAM expansion is optional.

Extensive instruction manual included.

Cartridge for VIC 20

HES

VIC FORTH is an exciting new cartridge for your VIC 20 computer. You now have a language that is more powerful than BASIC and easier to program than assembler! Some of VIC FORTH's major features are: ability to define your own words; this means a function not already supported can be created by you and added to VIC FORTH for future use. Full VIC sound and color capabilities are built into VIC FORTH. Names for your words may be up to 31 characters, unlike BASIC, which supports only 2 characters.

VIC FORTH comes with a superb full-screen editor which has 16 lines of 64 characters each (standard FORTH screen), using a horizontal scrolling window. VIC FORTH is an interactive language that is very memory efficient and much faster than BASIC.

VIC FORTH will run in a standard VIC 20, although extra memory is recommended. Programs can be loaded or saved from tape or disk. Character output can be sent to any device, including the VIC printer. Up to 24K additional bytes of memory can be automatically used in VIC FORTH. VIC FORTH is a nearly complete implementation of the "FORTH INTEREST GROUP" (fig) version of FORTH. The VIC FORTH editor follows closely the standard FORTH editor described in the book, "STARTING FORTH," plus it has many, many additional features.

71 Park Lane, Brisbane, CA 94005

Printed in U.S.A.

C301

HES

VIC FORTH

By Tom Zimmer

Instruction Manual

VIC FORTH

By Tom Zimmer

CONTENTS SUBJECT PAGE

1.0 INTRODUCTION

This manual describes the VICFORTH system for the COMMODORE VIC-20 computer. It is an extension of the FORTH Interest Group (fig) model of the language FORTH.

1.1 REQUIREMENTS

HARDWARE The VIC-20 must have at least 5K bytes of memory (which is available on a standard VIC-20),and VICFORTH will automatically adjust to more memory if you have it. The 3K ram expansion cartridge does not provide any additional memory for VICFORTH to use.

1.2 VICFORTH STARTUP

1. Verify the power is OFF!!.
 *****NEVER INSERT OR REMOVE A CARTRIDGE*****
 ***** WITH THE POWER ON !!!! *****
2. Plug the cartridge into the slot on the back.
3. ONLY THEN turn on the power.

The computer will start up with the VICFORTH sign on message in dark blue characters on a cyan background, with a green border. VICFORTH is now in control.

1.3 FIRST IMPRESSIONS

After VICFORTH is started as described previously, the FORTH operating system is running, with its compiler, interpreter, and text editor. If you press the RETURN key a few times, you will notice an 'OK' is printed after each RETURN is pressed. FORTH is just telling you that it recognized your request to do nothing, and has returned for your next command. In FORTH a line of input text is simply a line of commands, each separated by a space, which FORTH recognizes as command separator. Each command in the line is performed from left to right. If all commands are performed without encountering an error, then FORTH returns with the 'OK' prompt to tell you its mission was accomplished. If an error is encountered at any time during execution, then all processing stops, FORTH will issue an error message and wait for further instructions.

1.4 WHAT IS VICFORTH?

VICFORTH is an implementation of fig-FORTH with the addition of several words to interface to the VIC's sound and color capabilities. Over 250 WORDs are included in VICFORTH, and since FORTH is extensible, you can add many of your own commands to VICFORTH. If you have not used the FORTH language before, you will probably find FORTH's syntax to be somewhat strange. But don't let that bother you; the FORTH programming environment has been honed over a period of 12 years into a very efficient system. In BASIC where some arithmetic operators have different precedences, it is sometimes hard to remember which functions are performed first. To alleviate this problem BASIC uses parentheses to specify the order of operation. FORTH does not need to worry about such things, it uses a straight left to right process.

VICFORTH also contains words to allow you to control the character, border, and background colors of the screen.

Five sound control words are provided to allow manipulating the sound output in the VIC-20.

All I/O and several other words are VECTORED, to allow their function to be changed at runtime. This will allow you to drive a printer or other device using I/O that the system may not know exists. As an example, a driver routine could be written to drive a parallel printer over the user port.

In FORTH the term 'WORD' refers to an identifiable function or command, which in some computer languages is referred to as a subroutine or procedure.

In VICFORTH WORDS may be any length from one to thirtyone characters in length. This allows very descriptive names to be used in writing your programs. FORTH programs can then be much easier to read, than basic. It is important to note however that program readability is the responsibility of the person writing the program, and it is just as easy to write programs with all single character names, thus making them almost impossible to decipher. The use of readable names for your VICFORTH words is highly recommended.

1.5 NEW USER OF FORTH OR A VETERAN

We wish to state very clearly that this manual is NOT A FORTH BEGINNERS MANUAL!!. If you are a newcomer to the FORTH world, VICFORTH provides all of the program tools you will need to learn the FORTH language, BUT there are several books which you should purchase that will better lead you in learning the FORTH language. These books are as follows:

1. STARTING FORTH
2. fig-FORTH INSTALLATION MANUAL
3. fig-FORTH 6502 ASSEMBLY SOURCE LISTING
4. FORTH-79 STANDARD CONVERSION

The first two are highly recommended, and will provide you with most of the needed tutorial information. All of these manuals are available from the FORTH INTEREST GROUP.

Another point worth stressing is that the STARTING FORTH manual listed above is the best tutorial manual available today, BUT it describes a version of FORTH called FORTH-79, which is not identical to fig-FORTH/VICFORTH. So you should read the section of this manual called FORTH-79 DIFFERENCES while using the book, to assist you in learning FORTH. Also, the publication FORTH-79 STANDARD CONVERSION will be helpful in running the more complex examples in the STARTING FORTH book.

1.6 WHAT IS THE FORTH INTEREST GROUP

The FORTH INTEREST GROUP is an independent group of FORTH enthusiasts whose aim is to educate others, answer technical questions and to promote FORTH.

They may be contact at:

PO Box 1105	Phone for orders:
San Carlos, CA 94070	(415) 962-8653

They publish a newsletter FORTH DIMENSIONS ($15/yr) and have many other publications available.

2.0 VICFORTH SYSTEM CONFIGURATION

The following describes some of the aspects of how the VICFORTH system is organized and how it is similar and/or dissimilar to other FORTH systems.

NOTICE!!! If you are a complete beginner to FORTH, you should probably skip this section for now, and go to the book STARTING FORTH which you should have purchased. Just start working through the book. Do not be afraid of experimenting! The program in the cartridge can not be damaged by any programming error. Just press RUN/STOP RESTORE to restart if required.

NOTATION The following symbol terminology is used throughout this manual. The £ sign is used in place of the # sign, ie the one above the 3 key.

SYMBOLS	MEANING
al or addrl	16 bit address
n, nl, or n2	16 bit signed number
dl, d2	32 bit signed double number
ul, u2	16 bit unsigned number
bl	8 bit byte
c, cl	7 bit ascii character
f, fl	Boolean flag

tf	True boolean flag -- 1
ff	False boolean flag -- 0
string, t	Ascii text string
<sp>	Space character
<return>	The RETURN key

STACK VALUES (b --- t ; a)

b The stack before the word executes.
--- The word being executed.
t The ascii string which follows the word in some cases.
; Denotes the place where the <return> key would be pressed.
a The stack after the word executes.

STACK PARAMETER DESCRIPTIONS

n1/n2/n3

The value n1 was placed on the stack first, then n2, then n3. The notation is read "n1 under n2 under n3". n3 is on top of the stack.

EXAMPLE:

In the description of a FORTH word called <example>, the following might appear:

a1/n1/n2 --- t ; a2

This would be interpreted as follows: Before the word <example> is executed there are three items on the stack with n2 being the last placed on, n1 being the previous one, and a1 being the earliest one. The '---' represents the word <example> being executed, where 't' is text which must follow <example> before <return> is pressed at the ';' symbol. After the execution of the word <example>, the stack is left with one item 'a2'.

2.1 MEMORY ALLOCATION

VICFORTH will run on any COMMODORE VIC-20 computer with 5K to 29K bytes of user read/write (ram) memory. About 2K bytes of the available ram is used by the system, (FORTH & the KERNEL) so in a 5K VIC-20 this leaves about 3K bytes for the users program. The available ram is then divided into two segments. The first segment is screen/block buffer space; this area holds the source text for any program you enter. The second segment is called the dictionary; it holds the object code of any program you have compiled.

On a 5K VIC, VICFORTH will initialize itself with 2K bytes allotted to screen buffer space (which is equal to two screens), and 1K bytes for dictionary space. Since a compiled program is generally much smaller than the source which created it, a two screen (2K) source program should compile into 1K of dictionary space without difficulty.

There can, however, occur situations where you would like to have more than 1K of dictionary space. If this occurs, you can resegment the user ram with the system word "£BLOCKS". This word allows you to specify how many screens of buffer space you will have in the system. Any reduction in screen buffer space results in an increase in dictionary or compiled program space. The following command will allot one screen buffer, and return an additional 1024 bytes of ram to the dictionary.

```
1 <sp> £BLOCKS <return>
```

The £BLOCKS command clears out any program in the dictionary before resegmenting the user ram, so any program already compiled will be lost, effectively a COLD start is performed. The £BLOCKS command is normally executed right after powerup.

If your VIC-20 has more than 5K bytes of ram (excluding the 3K ram cartridge, which VICFORTH cannot access), VICFORTH will automatically expand the number of screen buffers up to six screens. A screen in VICFORTH is 1024 characters, or bytes of memory. In a fully expanded VIC-20 with 29K bytes of user memory, you may specify up to 26 screen buffers, to allow editing very large programs.

The command £BLOCKS always truncates the number of screens you request to within the range one to the maximum that memory will allow minus one; this assures there will always be at least 1K bytes of dictionary space.

2.1.1 STACK SPACE

The system stacks are located near the bottom of memory. The DATA stack is located in page zero , from about $60 (numbers preceded by $, like $60, indicate a hexadecimal number) down to $10, giving room for about 40 data stack entries. The RETURN stack is located in page one, the 6502 hardware stack. It shares this page with the terminal input buffer, and is thus limited to about 60 decimal levels of nesting. These stacks should be large enough for any properly developed program.

2.2 BLOCK INPUT/OUTPUT

The word BLOCK in the language FORTH provides the user with a method of accessing a very large data or program storage area typically on disk, as if all of the storage area were in the user ram memory space. In VICFORTH where a disk is typically not available, this process is simulated by allocating an area of user memory for the virtual screen buffers, and limiting the range of the area a user can access to the amount of screen buffer space currently specified. This technique allows many of FORTH's virtual memory operations to be performed normally, but within the restriction of user memory. The BLOCK command is a member of the list of vectored words in VICFORTH, and it's definition can be changed by the user.

In a disk based FORTH system the word BLOCK just described would perform the reads and writes to disk automatically. In this cassette-based system however, screens are read or written to cassette by the user using the READ and WRITE words described later in this chapter.

2.3 SYSTEM CALLS

COMMODORE has built into the VIC-20 a very powerful KERNEL. In BASIC, you do not have access to the KERNEL , since it must be passed several parameters in the machine registers to control the operation of a KERNEL call. In VICFORTH, however, a word 'SYS' has been provided which allows you to set all of the machine's registers including the status of the CARRY FLAG, before performing a system call. Once the call returns, the contents of all CPU registers is placed on FORTH's stack, and is available to you. This very versatile word allows you to access all

KERNEL functions from high level FORTH.

SYS (<f>/n1/n2/n3/a1 --- <f>/n4/n5/n6)
The SYS command allows calling any assembly language routine. 'f' is an optional CARRY set/reset flag; it is returned unmodified. n1, n2, and n3 are the A,X,Y registers respectively, and return their routine modified contents. a1 is the call address. Any called routine must return with an assembly 'RTS' instruction, and must not destroy the hardware stack contents. Here is an example of a system call to the VIC KERNEL:

```
          HEX
          : MESSAGES.OFF   0 0 0 FF90 SYS 3DROP ;
```

This word will turn off all KERNEL messages, for errors, and warnings, etc. VICFORTH is initialized with all messages ON. The "0 0 0" in the above definition are the manditory "A,X,Y" register values. The address "FF90" is the call address for the system call we are performing. The word SYS performs the system call, and the 3DROP after SYS removes the returned values of the "A,X,Y" registers after the call is performed, off the stack.

Here is a list of predefined system calls available in VICFORTH.

SETFLS (n1/n2/n3 ---) Set system logical file command, must be sent n1, the user assigned fileL, n2 the device number assigned to the fileL, and n3 the command to be sent to the device. See the " VIC PROGRAMMERS REFERENCE MANUAL".

OPEN (---) Open the device and fileL specified in the previous SETFLS command.

SAVE (a1/n1 ---) Save to cassette from memory at address a1, for a count of n1 bytes, as the file last specified by SETFLS.

SLOAD (a1 ---) Load from cassette to memory at address a1, the file last specified by SETFLS. The load buffer area must be large enough for the file being loaded.

CHKOUT (n1 ---) Send all character output to device n1. Device n1 must have been previously opened.

CLALL (---) Clear all channels and devices, restore all I/O to video and keyboard.

SETNAM (a1/n1 ---) Set the name at address 'a1', as the current file name, for a length of 'n1' characters. See also section 7.0, VICFORTH AND DISK.

2.4 CASSETTE INTERFACE

Five words have been included in the system to permit easy writing and reading of data to and from cassette. These are:

READ	(n1 ---)
READS	(n1/n2 ---)
WRITE	(n1 ---)
WRITES	(n1/n2 ---)

In these words, n1 is the first SCREEN or BLOCK to read or write. n2, when present, is the number of SCREENS or BLOCKS to read or write.

DON'T FORGET TO SETUP THE CASSETTE RECORDER!

The fifth word is -

LOADS (n1 ---)

n1 screens will be loaded from cassette, and compiled or interpreted. Each SCREEN will be read into BUFFER Ł1, and then a 1 LOAD is performed. Then the next SCREEN is read and loaded, etc. This procedure was chosen so that only a single buffer would be required, in order to have the system function with minimal memory. This means that very large programs may be loaded. The number n1 specifies the number of screens to be read and loaded, and it must match the actual number of screens on the cassette.

Two additional words direct the device number, and the file number to be used with READ and WRITE operations. These are "DŁ" and "FŁ". They are initialized to 1 by VRESET, for normal cassette operation. If you wish to direct READ/WRITE I/O to a device other than cassette, you will need to change the contents of these variables. See also section 7.0 on VICFORTH DISK OPERATION.

2.5 DOWN-LOADED WORDS

Two words in VICFORTH are DOWN-LOADED to RAM from ROM. They are:

FORTH and EDITOR

These two words are what are called VOCABULARIES. They contain a variable which changes as new words are added to the dictionary. For these variables to function correctly they must be in RAM, so both of these words are DOWN-LOADED. The total code that is moved is less than 60 bytes.

2.6 WHAT IS MISSING

There are several fig-FORTH words which are not in this system. For the most part these are words which are used for the DISK interface. They are not needed in this implementation. They are:

```
 BUFFER  +BUF    R/W          INDEX   TRIAD   PREV   USE
 FLUSH   UPDATE  EMPTY-BUFFERS        (ABORT) WARNING
```

If desired, an EXPERIENCED PROGRAMMER can add these and change the VECTOR for BLOCK.

2.7 NEXT

For those interested assembly language programmers, NEXT is located with the following code sequence:

```
          ' (LOOP) NFA 2 - @  CONSTANT  NEXT
```

Here a constant has been created with the name NEXT for future program reference.

2.8 MEMORY MAP

```
-----------------------------------------$0000
!           FORTHS DATA STACK           !
-----------------------------------------$0060 S0
!          VIC SYSTEM DATA AREA         !
-----------------------------------------$0100 TIB
!        TIB      /      RETURN STACK   !
```

```
-----------------------------------------$01FF RO
!         VIC WORKING STORAGE AREA       !
-----------------------------------------$0400
!            3K EXPANSION AREA           !
!          (not used by VICFORTH)        !
-----------------------------------------$1000
!                                        !SCREEN
!     SCREEN RAM IN EXPANDED SYSTEM      !
-----------------------------------------$1220
!         DOWN LOADED FORTH WORDS        !
-----------------------------------------$1260 UPO
!         USER TABLE AND VECTORS         !
-----------------------------------------$1340
!       FIRST SCREEN/BLOCK BUFFER        !FIRST
-----------------------------------------
!                                        !
!      ADDITIONAL OPTIONAL BUFFERS       !
!      UP TO THE MAX ALLOWED BY MEM      !
!                                        !
----------------------------------------- DPO &
!                                        ! LIMIT
!                                        ! 5K=
!        USER DICTIONARY RAM AREA        ! $1B48
!                                        ! 13K=
!                                        ! $2B56
----------------------------------------- EM($283)
```

Memory end is specified in variable EM

```
-----------------------------------------$A000
!                                        !
!        VICFORTH KERNEL OBJECT          !
!                                        !
-----------------------------------------$BFFF
```

3.0 EDITOR

3.1 EDITOR COMMENTS

The EDITOR in VICFORTH is modeled after the editor described in the introductory book STARTING FORTH by L. Brodie.

Note: For the examples in the book STARTING FORTH and the discussion in this manual, use screens numbered 1 to 6 on an 8K expanded system, or screens 1 and 2 on a nonexpanded system.

You can now start the edit session by entering the following:

```
          n1 <sp> EDIT <return>
```

The EDITOR vocabulary is selected and the BACKGROUND color has switched to WHITE, indicating the editor has been selected. The screen will be split in two - the top 16 lines are used for entering text (edit area) and the bottom 6 for entering edit commands (command area). If you just turned on the computer, you will have 16 lines of garbage at the top of the screen. To clear out the edit buffer in preparation for editing, type:

```
          WIPE    <return>
```

This will clear the edit screen to all spaces. While in the editor, the window around the cursor is always displayed just before the next command line is fetched from the keyboard, so you always have an updated view of what your edit commands are doing to the text. You will also notice two double-digit numbers displayed in the lower right corner of the edit window. The rightmost number is the column position of the cursor in the edit screen, the left number is the number of the current edit screen. The cursor keys are also redirected in the editor, so they allow you to scan around through the edit screen with very little effort. To leave the EDIT mode, just press RUN/STOP RESTORE, and the edit mode will be terminated.

For a complete discussion of the use of the editor commands you should refer to the book STARTING FORTH, although it is similar enough to a typical fig-FORTH editor that an experienced FORTH programmer should be able to use the editor with just the abbreviated discussion presented in the next section.

Note: Three characters are viewed incorrectly in the edit window.

[is ┼   ] is │   @ is ─

3.2 AN EDITOR EXAMPLE

This editor contains both Screen and Line edit commands. The line edit commands have been taken directly from STARTING FORTH, and are described in that book. The screen edit commands give you the ability to see the form of the text as it is entered into the edit screen. To enter the screen edit mode, press the <shift> key, and the <INST/DEL> key. The border of the screen will switch to yellow (lighter grey on a b/w TV), and is now waiting for you to type any text you wish inserted into the screen. Type the following:

```
   THIS IS VICFORTH <F5>
```

The text THIS IS VICFORTH was inserted in line zero of the screen, and when you pressed <F5> (F5 is the tan function key labeled F5), the cursor moved to the beginning of the next line. The <F5> function key is like <return>, in that it moves the cursor to the next line, but it leaves you in the INSERT mode. If you accidentally pressed <return>, you will have noticed the border switch back to GREEN, indicating you have left the INSERT mode. To repeat, you must press <return> to exit the INSERT mode, at which time the border will switch back to the normal GREEN. While in the INSERT mode, the <del> key is enabled to delete characters before the cursor in the edit window, rather than characters on the command lines at the bottom of the screen.

3.2.1 AN EDITOR EXAMPLE (contd)

Type in the following edit commands in the edit mode to get a feel for the different edit commands.

```
1 EDIT
WIPE
3 T
P THIS IS A LINE OF TEEXT
K
P THIS IS A LINE OF TEXT.
U HERE IS ANOTHER ONE
3 T
F TEXT
E
5 T
P THIS IS AANOTHER ONE ONE.
F AANOTHER
R ANOTHER LINE
D ONE
```

```
          1 DEL
          4 T
          F A
          TILL ER
          8 T
          P HERE IS A TEST LIST
          F LIST
          E
          8 T
          D TEST
          I NEW LINE
          3 T P EXAMPLE OF TWO COMMANDS
          X
```

To leave the editor, type one of the following:

FORTH	(Terminate the session)
VRESET	(Reset all I/O vectors)

--- or ---

hit RUN/STOP & RESTORE (Reselects FORTH and resets I/O vectors)

3.3 EDITOR WORDS

LINE EDITING COMMANDS From STARTING FORTH editor:

WORD FUNCTION

T (nl ---) Sets the edit pointer to the start of line nl.

P (--- <t> ;) Text following space after P is placed into line holding edit pointer.

U (--- <t> ;) Text following space after U is placed under the current line and all lower lines are moved down.

M (n1/n2 ---) Copies current edit pointer line UNDER line n2 in screen n1.

X (---) Deletes line containing the edit pointer and moves lower lines up one. Line £15 becomes blank. The line is held in PADI.

Added from fig editors:

H (---) Holds the edit pointer line in PADI.

K (---) Kills (erases) the edit pointer line.

S (---) Spreads the lines at the edit pointer. All lines from the edit pointer are moved down. Line 15 is lost.

TOP (---) Move edit pointer to TOP of screen.

NOTE: In (--- <t> ;), The <t> symbol indicates the text is optional. Typing <return> without any text will use the current contents of PADI, or PADF.

EDITING COMMANDS
From STARTING FORTH editor:

WORD	FUNCTION

F (--- <T> ;) Find first occurance of text following 'F'. Starts at current edit pointer.

E (---) Erases as many characters going backward as the length of the last 'F' command.

D (---) Deletes the first occurance of text following the D command, searches from edit pointer till end of screen.

TILL (--- <t> ;) Deletes all text starting at edit pointer until and including the string following the command TILL. Works on current LINE only. If string is not found, no delete occurs.

I (--- <t> ;) Inserts text following the command I into the edit buffer at the current position of the edit pointer. Text following that is too long for the line is lost.

R (--- <t> ;) Replaces the string just found by 'F' with the string following the R command.

Additions to the editor:

DEL (nl ---) DELetes nl characters BEFORE the edit pointer, and compresses the line to omit the space.

C (nl ---) Move the cursor by the signed amount nl characters (positive for forward move, negative for backward move). This word also redisplays the current edit window.

N (---) Move the edit pointer to the top of the next higher screen buffer. Limited by BMAX.

B (---) Move the edit pointer to the top of the previous (lower) screen buffer. Limited by

BMAX.

EDIT (n1 ---) Selects the edit mode, with n1 as the screen to be edited. Moves the edit pointer to the top of screen n1. Revectors CR & KEY to show the current edit window, and make cursor keys functional.

MISCELLANEOUS EDITOR WORDS

These words are not normally used from the keyboard. They are provided to allow editor expansion.

(F) (---) Search for text in PADF till end of screen.

(I) (---) Insert the current contents of PADI into the edit buffer at the cursor. Text too long for the line is lost.

PADF (--- a1) Returns address of find buffer.

PADI (--- a1) Returns address of insert buffer.

PAD (--- a1) Returns address of scratchPAD area a1.

TEXT (c1 --- t ;) Accepts the text following the command TEXT into the scratchPAD area until the character with ASCII value c1.

GTEXT (a1 ---) Accepts text from input stream until a delimiting ↑ is found, or the <enter> key is pressed. The text is placed at address a1.

!CUR (n1 ---) Sets the edit pointer to value nl. (nl is limited to 0 <= nl <= 1023.)

Additional immediate key functions:

Cursor keys Active, allow scanning through the edit buffer character by character, or with auto repeat.

HOME key Moves the cursor to the top of the edit screen.

Function keys:

F1 Tab cursor right 4 characters.

F3 Tab cursor left 4 characters.

F5 Move cursor to beginning of next line.

F2 Move edit to the Next higher screen number.

F4 Move edit Back to the previous screen number.

F7 Find the next occurance of search string given by the last 'F' command and leave insert mode if not found.

F8 Replace the most recently found string with the text specified in the most recent 'R' command.

<INST> This key enables the INSERT mode, the BORDER color is changed to YELLOW to indicate the insert mode, and all keys are inserted into the edit screen as they are typed. The <DEL> key is also enabled, to delete characters on the edit screen preceeding the cursor. The <return> key leaves the INSERT mode, and the BORDER color returns to GREEN.

4.0 VICFORTH SPECIFICS

4.1 COLOR CONTROL

VICFORTH provides you with words to control the background, border, and character color of the VIC-20, without having to poke into memory using an obscure calculation. The background is controlled as follows:

```
nl <sp> BGROUND <return>
```

Where nl is a number in the range 0 to 15, giving 16 possible background colors. The border is controlled as follows:

```
nl <sp> BORDER <return>
```

Where nl is in the range 0 to 7, giving 8 possible border colors.The border colors are the same as the character colors, listed on the front of the top row of keys. Just subtract one from the key number to select the proper color. EXAMPLE:

```
7 <sp> BORDER <return>
```

The above command will select a YELLOW BORDER color.

The character color is selected in the same manner as BASIC. Press the CTRL key, and the color key (1-8), to select the character color you desire. The color key pressed is also placed in the command stream, although it is not a printable character. To select the color of characters to be printed during program execution, first find out what the KEY value is for the color desired, as follows:

```
   KEY <return>
   ↑2       ( the up arrow two is CTRL 2)
   . <return>  ( will print the value of the key )
   5        ( printed by the computer as the value )
            ( of the control 2 key. )
```

Now all that is required since we know the value of the key to select the color we want, is to EMIT it:

```
  5 EMIT <return> ( this will select WHITE
characters, )
  ( this can be put in a program too!   )
```

To switch colors to another color, simply EMIT the value for that color.

Color word glossary:

BGROUND (nl ---) The value nl in the range 0 to 15 sets the BackGROUND color of the screen. Refer to the VIC-20 Users Manual for color value selection.

BORDER (nl ---) The value nl in the range 0 to 7 sets the BORDER color. The value of nl is the same as the keyboard keys 1-8, minus one 1, i.e., 7 is YELLOW and 2 is RED.

EMIT (nl ---) Sends ASCII value nl to the current output device (usually the screen). See Appendix J of the VIC User Manual for possible values. The values to EMIT to select the different character colors are:

BLK = 144	WHT = 5	RED = 28
PUR = 156	GRN = 30	BLU = 31

```
        CYN = 159      YEL = 158
```

By the way, the SCREEN in VICFORTH is always located at $1000, regardless of how much memory your system has.

4.2 SOUND CONTROL

Words have been included to control the VOLUME and frequency of all four of the VIC-20's voices. The VOLUME is controlled as follows:

```
        n1 <sp> VOLUME <return>
```

The value of n1 is used to select the volume of all voices currently active. Values 0 to 15 are valid.

The frequency of the four voices are controlled by storing values into the name of the voice to be turned on. The names are

```
        ALTO     TENOR      SOPRANO      NOISE
```

The voice is turned on by storing a value greater than 128 decimal into the desired voice. Values 128 to 254 generate increasingly higher frequencies, while a value of 255 will generate the lowest frequency for a given voice. To generate a middle C on the TENOR voice, enter the following:

```
        10 <sp> VOLUME <return>
        195 <sp> TENOR <sp> C! <return>
```

If the volume is turned up on your television, you will hear a note being generated. To turn OFF the voice, simply store a zero into it:

```
        0 <sp> TENOR <sp> C! <return>
```

The voice will be silenced.

Sound control glossary:

VOLUME (n1 ---) Sets the VOLUME of all voices as the value of n1. n1 is in the range 0 to 15. 15 is loudest and 0 turns all voices off.

ALTO
TENOR
SOPRANO
NOISE (--- a1) Constants which return the address a1 of the tone voices of the VIC-20. Each voice is programmed for frequency by storing an 8 bit value in the range 128 to 255 into its address.
EXAMPLE:

```
            195 <sp> TENOR <sp> C! <return>
```

The above will turn on the middle voice of the VIC-20. Any value less than 128 will turn off the voice.

4.3 USER PORT

COMMODORE has provided a full 8 bit port in the VIC-20, and we have included a constant which returns the address of the DATA register of that port. The name of the word is UPORT, and it can be used to access 8 bits of external data. This port is initialized as an INPUT port, so any TTL level can be observed as follows:

```
          UPORT <sp> C@ <return>
```

This will place the value of the data on the port on the FORTH data stack. Once the data is on the stack, it can be printed, or manipulated in many ways. If you wished to use the UPORT as an OUTPUT port, you will have to initialize UPORT as follows:

```
255 <sp> UPORT <sp> 2+ <sp> C! <return>
```

This stores a binary value of all ones into the UPORT data direction register of the 6522 VIA chip in the computer. The VIA is designed to make any data bits output, when the corresponding bits have been set to one in the data direction register.

To control the output data from the port:

```
n1 <sp> UPORT <sp> C! <return>
```

Where n1 is a number in the range 0 to 255, which is to be sent to the user port.

User port glossary:

UPORT (--- a1) Returns the address a1 of the user port.

4.4 PRINTER OUTPUT

VICFORTH includes words to allow data or programs to be listed to the serial printer. The word PRINT allows any character output from the line of text following PRINT to be sent to the serial printer, then after that line is interpreted, output returns to the video screen. A lower level word PRON turns on the printer port, and leaves all character output going to the serial printer, till a CLALL (clear all) command it executed. These commands would be used within your program to control printer output.

To list a screen to the printer:

```
  PRINT nl LIST
```

where nl is the desired screen number.

Since all I/O is vectored in VICFORTH, it is also possible to write a printer driver for other types of printers than are supported by the VIC KERNEL. Vectored I/O will be discussed in a later section.

Printer output glossary:

PRON (---) Send all character output to the printer port.

CLALL (---) Restore all KERNEL I/O to their default values - keyboard and video.

PRINT (--- t ;) Any character output from the line of text following the PRINT command, will be channeled to the printer port. When the line completes interpretation, output will be restored to the video.

NOTE: If an ERROR is encountered in the PRINT command line, all character output will remain on the printer. To restore character output to the video screen, type:

```
      CLALL <return>
```

This will clear the print buffer, and return character output to the video screen.

NOTE: DO NOT USE THE PRINT WORD FROM WITHIN THE EDITOR !! The CR word is revectored by the editor, and will cause your listings to be printed without carriage returns.

4.5 SCREEN BUFFER CONTROL

Memory is a precious commodity in a small computer, and it needs to be used efficiently. In line with this, VICFORTH includes a method of reallocating use memory such that efficient use can be made of the limited resources available. Three words are provided to control memory usage. They are:

```
        BMAX EMPTY £BLOCKS
```

BMAX is a user variable, which contains the highest block number the system will allow you to access. It is initialized at COLD start to a value of six on a 13K system, and two on a 5K system. Due to memory restrictions a 5K system is limited to 2 screen buffers. The following applies to larger machines. A 13K machine can have up to 10 screen buffers, which leaves very little for program compilation, but which allows large programs to be edited. To select a different number of screen buffers than is set at COLD start, perform the following:

```
    7 <sp> £BLOCKS <return>
```

The system now is configured for seven screen buffers, with the dictionary space reduced accordingly to four. £BLOCKS first sets the value of BMAX, after limiting it to the amount of memory you have, then calls EMPTY to reset memory, and empty out the dictionary of any current program.

The word EMPTY can also be used from the keyboard, when you have been experimenting with some new definitions, and you would like to clean out the dictionary.

Screen buffer control glossary:

BMAX (--- a1) A user variable which contains the highest block the user can currently access.

EMPTY (---) Resets the VOCABULARY pointers to their COLD start values, and resets the dictionary pointer to LIMIT.

£BLOCKS (n1 ---) Select n1 as the current number of screen buffers for the system. Limits n1 to the range 1 to max that memory will allow.

4.6 VLIST

The word VLIST in VICFORTH as in fig-FORTH gives you a list of all of the commands currently available. The word is executed by simply typing its name:

```
VLIST <return>
```

You will then see a vertical list of words, each having an address to its left. The address is the Parameter field address of the word, and can be useful in experimenting with VICFORTH's internals. If you press any character key on the keyboard, the scrolling list of words will pause until another key is pressed or RUN/STOP is pressed to cause the VLIST to stop.

4.7 ADDITIONAL UTILITIES

Here are some additional utility definitions contained in VICFORTH.

DUMP (a1/n1 ---) This utility is provided to allow you to dump the contents of memory to the screen in a byte format. The contents of memory are printed in the current base, with four numbers per line and an address to the left. An example follows:

```
HEX <return>
A000 8 DUMP <return>
A000  XX  XX  XX  XX
A004  XX  XX  XX  XX
```

Where a1 is the starting address of the DUMP, and n1 is the number of characters to display.

THRU (n1/n2 ---) Screens numbered n1 through screen n2 are all loaded in sequence, from the memory buffers. These screens must have all been previously read in from cassette.

ASCII (--- t ;) The first letter of the text word following will be placed on the stack when in the interpret mode, or compiled into the dictionary if in the compile mode.

EM (--- a1) This is a system variable, which returns the address in memory where the end of memory pointer resides. Here is a simple definition to calculate the amount of free memory in the dictionary and print that value:

```
          : .FREE  EM @ HERE - . ;
```

U.   (n1 ---)   Prints the value of n1 unsigned.

H.   (n1 ---)   Prints the value of n1 as an unsigned hexdecimal value.

5.0 VECTORS

5.1 WHAT ARE THEY?

As you may have learned by now, FORTH is a macro language, that is, FORTH is made up of many simple words written in assembly language. These are used to create powerful words, by stringing several smaller words together in a line. This simple technique makes FORTH a very powerful tool in writing programs. There is one small problem with this method though - there are times when a low level word needs to perform a slightly modified function, so a higher level word can also do something a little differently. As an example, suppose you had a parallel printer. Since the VIC-20 has an 8 bit user port, it would be nice to be able to send all of your listings to that port, rather than the video screen. The only problem with this is that the VIC-20 KERNEL doesn't know about your printer on the USER port, and can not easily be made to talk to it. In VICFORTH this will not be a major problem. All I/O in VICFORTH is vectored, so you can make all character output from FORTH go to your own routine rather than just the routines that the VIC-20 already knows about.

5.2 VECTORED WORDS IN VICFORTH

VICFORTH has included a minimum set of words which should fill the needs of most users wishing to revector various operations in FORTH. Here is a list of all of the VECTORed words in VICFORTH, and their positions in the two vector tables, I/O, and VWORDS.

Position		Word
HEX	DECIMAL	
00	00	KEY
02	02	EMIT
04	04	?TERMINAL
06	06	CR
08	08	CREATE
0A	10	NUMBER
0C	12	ERROR
0E	14	. (dot)
10	16	-FIND
12	18	MESSAGE
14	20	BLOCK
16	22	EXPECT
18	24	CTBL

Note: User created vectored words start at an index of 30, and may consist of up to 20 total vectors, two bytes each.

5.3 VECTOR CONTROL

Here is an example of how to revector the character output to a different device than the video screen.

We will give an example of a printer driver for the USER port, for a seven (7) bit printer driver, with the eighth (8th) bit used as a strobe.

The first thing we need is a word to initialize the port for all 8 bits as outputs:

```
( --- )              ( initialize the user port )
: PINIT        255 UPORT 2+ C! O UPORT C! ;
```

Next we need a word to send characters to the user port and strobe bit 8 of the port:

```
( --- )          ( Strobes bit 8 of user port )
: STROBE8 UPORT 128 TOGGLE UPORT 128 TOGGLE ;

( nl --- )       ( Send char to user port )
: PORTOUT 127 AND UPORT C! STROBE8 ;
```

Now we have a routine to send characters to the USER port, all we have to do is vector EMIT to the new character output word.

```
        : TO.UPORT   ' PORTOUT CFA  I/O 2+ ! ;
```

The routine above takes the CFA (code field address) of the new driver word, and stores it into the third and fourth bytes of the I/O vector table in user memory. The instant this has been done, any further character output will be going to the new driver routine. If it doesn't work, then the program may hang. To restore the I/O vectors to their initial values hit RUN/STOP & RESTORE or type VRESET. Either of these will restore all of the VECTORS to their initial values.

NOTE: The example above was to show how to revector

EMIT. It was not an example of a complete printer driver, since most printers require handshaking that I did not include. You will have to study the data sheets on the 6522 chip for a while before attempting such a driver.

Vector Word Glossary

VECTOR (n1 --- t ;) This is a new defining word, used to add additional words to VICFORTH that are to be vectored. It is used as follows:

```
        30 <sp> VECTOR <vector name>
```

After the above is executed, any time <vector name> is executed, it will execute the routine whose CFA is in I/O + 30. You must therefore store the CFA of a valid dictionary word into it before executing <vector name>.

Note: User VECTORs are from 30 to 50 decimal.

VWORDS (--- a1) This word returns the address a1 of the beginning of the initial value vector table in ROM. This data may be accessed in the same manner as the I/O table to obtain the actual routine's CFA for a given vectored word.

I/O (--- a1) This word returns the address a1 of the beginning of the user vector table in RAM. This table is used to change the function of selected dictionary words.

VRESET (---) This word when executed RESETS all of the vectored dictionary words to their initial values, by moving the data at VWORDS to the

RAM table at I/O.

6.0 ERRORS, CRASHES AND OTHER PROBLEMS

6.1 ERROR MESSAGES

VICFORTH is an 8K byte program. In an effort to include as many user features as possible, we have had to omit lengthy error messages. Here then is a list of the error numbers, and the error messages to go with them.

ERROR£		ERROR MESSAGE
DEC	HEX	
0	0	VICFORTH DOESN'T KNOW THIS WORD.
1	1	THE DATA STACK IS ALREADY EMPTY.
2	2	OUT OF USER MEMORY.
8	8	SCREEN BLOCK RANGE ERROR. You asked for an invalid screen£.
17	11	USE WHILE COMPILING ONLY. This word can't be used while executing.
18	12	USE DURING EXECUTION ONLY. This word can't be used while compiling.
19	13	CONDITIONALS NOT PAIRED. Match your IF - ELSE -ENDIFs..etc.
20	14	THIS DEFINITION IS NOT FINISHED. You started a conditional without completing it. IE: BEGIN missing UNTIL.

21 15 THIS WORD IN A PROTECTED DICTIONARY.
You can't forget anything below FENCE.

22 16 USE ONLY WHEN LOADING.

23 17 EDIT POINTER IS OFF CURRENT EDIT SCREEN.

24 18 DECLARE YOUR VOCABULARY.
Specify the VOCABULARY on which you wish to perform the operation in, IE: FORTH or EDITOR.
The VIC-20 KERNEL has several I/O errors that can occur, these will be printed out as follows.

I/0 ERROR £5

The above example indicates an attempt to talk to a device which is not in the system. like a disk, or printer. here is a full list of the errors that can be printed by the KERNEL. These message numbers are always printed in DECIMAL.

I/O ERROR £	COMMENT
0	ROUTINE TERMINATED BY STOP KEY.
1	TOO MANY OPEN FILES.
2	FILE ALREADY OPEN.
3	FILE NOT OPEN.
4	FILE NOT FOUND.
5	DEVICE NOT PRESENT.
6	FILE IS NOT AN INPUT FILE.
7	FILE IS NOT AN OUTPUT FILE.
8	FILE NAME IS MISSING.
9	ILLEGAL DEVICE NUMBER.

6.2 WHAT DO I DO WHEN IT CRASHES?

VICFORTH provides a much more powerful programming environment than BASIC. Unfortunately it also places a higher level of responsibility on you, the programmer, than BASIC. There are many ways to cause FORTH to GO AWAY! When this happens, there are several ways to recover. If the crash is relatively minor, you can press RUN/STOP RESTORE, and FORTH will sign back on as if nothing has happened. There are however some cases when this will not work. If this happens, turn off the power, and then turn it back on after a few seconds, and VICFORTH will restart. It is advisable to save any large program to cassette tape prior to attempting to execute it. This may save you a lot of time later.

6.3 MESSAGE

Message is vectored in VICFORTH, so although full length error messages are not built into the system, one can add full length error messages by revectoring MESSAGE to a user written procedure. Type the following lines into screen 1, but don't type in the line numbers.

```
0   DECIMAL                              (NEW MESSAGE ROUTINE)
1   : NEW MESSAGE CR CLALL               ( N1 --- )
2      DUP  0 = IF ." WHAT?"                          ENDIF
3      DUP  1 = IF ." STACK EMPTY!"                   ENDIF
4      DUP  2 = IF ." MEMORY SPACE EXHAUSTED"         ENDIF
5      DUP  8 = IF ." BLOCK RANGE ERROR"              ENDIF
6      DUP 17 = IF ." FOR COMPILING ONLY"             ENDIF
7      DUP 18 = IF ." FOR EXECUTING ONLY"             ENDIF
8      DUP 19 = IF ." IMPROPER CONDITIONALS"          ENDIF
9      DUP 20 = IF ." INCOMPLETE DEFINITION"          ENDIF
10     DUP 21 = IF ." THIS WORD PROTECTED"            ENDIF
11     DUP 22 = IF ." USE ONLY WHEN LOADING"          ENDIF
```

```
12     DUP 23 = IF ." CURSOR OFF SCREEN"       ENDIF
13     DUP 24 = IF ." SPECIFY THE VOCABULARY" ENDIF
14    SP! QUIT ;
15   : FULL ' NEW.MESSAGE CFA I/O 18 + ! ; FULL
DECIMAL ;S
```

The above definition will now be inserted into the VECTOR for MESSAGE, as shown in line 15 above. To reinstate the messages after any warm start type: FULL <return>.

6.4 FORTH-79 DIFFERENCES

VICFORTH is a fig-FORTH implementation of the FORTH language. The differences between VICFORTH and FORTH-79 will be covered here on a chapter by chapter basis of "STARTING FORTH" to help you understand and adjust to the difference.

STARTING FORTH comments

Pg. 12,13 Change the definition of MARGIN to:

```
: MARGIN CR 5 SPACES ;
```

The VIC-20 has a narrow screen. The above change will improve the appearance of the demo.

Pg. 60 The screens in VICFORTH are numbered 1 to 6. An unexpanded VIC-20 has screens 1 and 2.

Pg. 68 Remember that lines in VICFORTH longer than 22 characters will wrap around. VICFORTH still has 64 character lines internally.

Pg. 76 FLUSH This word is not needed in this cassette-based system.

Pg. 77 S The editor 'S' command is not included, due to space restriction. Use the 'F' (find) command. In VICFORTH the 'S' command is used for Spread a line, which makes room at the current edit line for additional text to be inserted.

Pg. 83 Here is the VICFORTH definition for a stack print utility:

```
HEX
60 VARIABLE S0
 : DEPTH SP@ S0 @ SWAP - 2 / ;
: .S  ?STACK CR DEPTH
      IF SP@ 2 - S0 @ 2 -
         DO I @  5 .R - 2
         +LOOP
     ELSE  ." EMPTY " ENDIF ;
```

Pg. 91 0> This operator not supported, use ' 0 > '. (leave a space)

Pg. 101 ?DUP Not supported use -DUP.
ABORT" not supported, use:

```
    IF ." ERROR MESSAGE " SP! QUIT ENDIF
```

Pg. 107 1- 2- 2* 2/ Not supported, use:
1 - 2 - 2 * 2 / (leave a space)

Pg. 110 I' and J are not supported, here are their definitions:

```
 :  I' R> R> R SWAP >R SWAP >R  ;
 :  J R> R> R> R SWAP >R SWAP >R SWAP >R ;
```

Pg. 143 PAGE Not supported, here is the definition:

DECIMAL : PAGE 147 EMIT ;

U.R Not supported, here is the definition:
: U.R 0 SWAP D.R ;

Pg. 153 2* and 2/ not supported. Use 2 * and 2 /.

Pg. 161 /LOOP Not supported in VICFORTH.

Pg. 164 DOUBLE NUMBER DELIMITERS. VICFORTH only recognizes the decimal point "." as a double number delimiter.

Pg. 173 Only D+, D.R supported, for DNEGATE use DMINUS.

PG. 174 Only M* and M/ supplied, but use the definitions in the fig-FORTH installation manual for these words.

Pg. 183 VARIABLE The definition of VARIABLE used in VICFORTH is the fig-FORTH definition, which requires an initial value to be on the stack before creating the variable.
EXAMPLE:

```
12 VARIABLE DATE
```

This will create a variable with an initial value of 12.

Pg. 193 2VARIABLE, 2CONSTANT, 2@, and 2! are not supplied in VICFORTH. See 79-STANDARD conversion.

Pg. 207 CREATE This word functions

differently in VICFORTH/fig-FORTH than in this book, use the following to create the definition of LIMITS as shown in the book.

```
   220 VARIABLE LIMITS 340 , 170 , 100 , 190 ,
```

NOTE: The rule of thumb is to use VARIABLE in place of CREATE for definitions which DONOT have DOES> in them. If the definition is of the form:

```
           CREATE    xxxx    DOES>    xxxx
then use: <BUILDS    xxxx    DOES>    xxxx
```

This conforms to the fig-FORTH usage.

Pg. 216 FIND and EXECUTE VICFORTH uses the fig-FORTH word -FIND in place of FIND. In fig-FORTH the word EXECUTE must receive the code field address (CFA) instead of the parameter field address (PFA). Change the example on this page as follows:

```
      ' GREET CFA EXECUTE <return>
```

FORTH responds with:

```
      HELLO I SPEAK FORTH OK
```

Pg. 217 VECTORED EXECUTION The techniques will work on VICFORTH with the modification that the addresses obtained with '(tick) are converted to code field addresses (CFA) by the use of CFA. Example, line 6 would read:

```
      ' HELLO CFA 'ALOHA !
```

SAY The definition of SAY in VICFORTH is:

```
      :SAY [COMPILE] ' CFA 'ALOHA ! ;
```

There are two changes here. The word ' (tick) is used, and because in fig-FORTH it is IMMEDIATE, it must be compiled by the [COMPILE] word. The second change is the use of CFA to prepare the address for EXECUTE.

Pg. 219 NUMBER This definition is vectored in VICFORTH but to revector NUMBER as shown on this page, in VICFORTH you must say:

```
    DECIMAL ' (number) I/O 10 + !
```

VICFORTH uses a table, where each entry in the table does not have to have a header. Again due to space restrictions.

['] This word is not supported in VICFORTH. Functionally it is the same as ' (tick).

Pg. 220 NAME LENGTHS VICFORTH supports full names up to 31 characters in length.

Pg. 230 EXIT In VICFORTH use ;S.

Pg. 232 RELOAD is not needed, all code is in ROM.

Pg. 233, 237 H In VICFORTH use DP.

Pg. 235 'S In VICFORTH use SP@.

Pg. 239 OPERATOR Not needed in VICFORTH.

Pg. 240 >IN Use IN in VICFORTH.
OFFSET Not needed in this cassette

VICFORTH.

Pg. 243 ASSEMBLER The assembler may be loaded from cassette, if required.

Pg. 245 LOCATE Not supported in VICFORTH.

Pg. 255-257 UPDATE, FLUSH, SAVE-BUFFERS, EMPTY-BUFFER, BUFFER These words are not in VICFORTH since it is cassette based.

Pg. 259 LABEL In VICFORTH, must change to:

```
  : LABEL 8 * ' "LABEL" 3 + + 8 TYPE SPACE ;
```

VICFORTH does not support ['] and the word ' (tick) serves the same function in a definition.

Pg. 261 >TYPE Not needed in VICFORTH, use TYPE.

Pg. 266 MOVE, <CMOVE Not in VICFORTH.

Pg. 272 H Use DP.

Pg. 281 -TEXT In VICFORTH use (MATCH), see fig-FORTH installation manual for its definition.

Pg. 291 VARIABLE, CREATE To create the STARTING FORTH type of definition for VARIABLE, use:

```
            : VARIABLE <BUILDS 2 ALLOT DOES> ;
```

To create the VICFORTH/fig-FORTH definition for VARIABLE, you do it this way:

```
            : VARIABLE <BUILDS , DOES> ;
```

The fig-FORTH word CREATE is used only for creating CODE word headers.

Pg. 292   DEFINING WORDS   The definition of a DEFINING-WORD in the book must be changed to:

```
  : DEFINING-WORD <BUILDS (compile-time action)
                  DOES>    (run-time  action) ;
```

The EXAMPLE for CONSTANT is then:

```
    :  CONSTANT <BUILDS , DOES> @ ;
```

PG. 297   ARRAY The definition of ARRAY must be changed to:

```
    : ARRAY <BUILDS OVER   ,   *   ALLOT
            DOES>  DUP  @   ROT    *  +  +   2+ ;
```

Pg. 313   DOES>   For most purposes VICFORTH is the same as FORTH-79, but for the advanced programmer, see the document FORTH-79 STANDARD CONVERSION from the FORTH INTEREST GROUP.

Pg. 332   JOB, 1FIELD, 2FIELD These must change to account for the different CREATE, use:

```
        20 VARIABLE JOB    24 ,
        00 VARIABLE 1FIELD 30 ,
        30 VARIABLE 2FIELD 12 ,
```

Pg. 339   SIMPLE FILES In screen 240 change the definitions as follows:

```
          00 VARIABLE SURNAME 16 ,
          16 VARIABLE GIVEN   12 ,
          28 VARIABLE JOB      24 ,
          52 VARIABLE PHONE    12 ,
```

FREE   Change the definition of FREE to the following:

```
:  FREE 1  MAXRECS 0
        DO I ŁRECORD ! RECORD C@ 33 <
                    IF 0= LEAVE ENDIF
        LOOP
        IF ." FILE FULL " QUIT ENDIF ;
```

Pg. 339   ' (tick)   Prefix all occurances of ' with the word [COMPILE], EXAMPLE:

```
      : CHANGE [COMPILE] ' PUT ;
```

Pg. 347   DENSITY, THETA, STRING   Prefix all of the words when defined with a zero (0).

7.0 VICFORTH AND DISK

The VICFORTH system as supplied to you is specifically designed to work with cassette, but since some people are likely to want to use the VIC-20 with the 1540 disk drives, here are some routines that will allow you to save programs on the disk drive.

```
DECIMAL  (select the decimal number base)

        (select the CASSETTE for further operation)
:  CASSETTE 0 0 SETNAM 1 DŁ ! 1 FŁ ! ;

        (Select the DISK for further operation)
:  DISK 0 0  SETNAM 8 DŁ ! 15 FŁ ! ;

        (Write screen nl to the name following
NWRITE)
         (nl --- t ;)
:  NWRITE NAME WRITE ;

         (Read screen nl with name following NREAD)
         (nl --- t ;)
:  NREAD NAME READ ;
```

Usage of these routines is very simple. Enter them into a screen and save them on cassette for later use. When you wish to load or save screens to disk, load in these routines from cassette, put a formatted disk into the 1540 disk drive.

Load a created file from cassette, or create a file you wish to send to disk, then perform the following;

```
         DISK <return>
```

This selects all READ and WRITE operations to go to the DISK.

```
         1 <sp> NWRITE <sp> TEST1 <return>
```

The above command will write screen 1 to disk with the name TEST1. All files on disk must have names, although they can have numbers for names.

When you wish to read TEST1 back into a screen enter the following command line.

```
         2 <sp> NREAD <sp> TEST1 <return>
```

TEST1 will be read into screen 2. To rewrite TEST1 back to the same filename, you must prefix the name with the "@" sign as follows:

```
          2 <sp> NWRITE <sp> @TEST1 <return>
```

Screen 2 will be written back to disk, overwriting the original contents of file TEST1.

To reselect the cassette for further I/O operations execute the following command;

```
          CASSETTE <return>
```

The cassette device will be reselected, and the filename will be reset to NO-NAME as is normal for cassette block I/O.

A disk to be used with VICFORTH can be initialized using the normal built in BASIC technique.

NAME HANDLING GLOSSARY

FN (--- a1) A user variable which returns a pointer to the beginning of a 64 byte buffer, used to hold the current FILENAME.

SETNAM (a1/n1 ---) See SYSTEM CALLS section 2.3.

NAME (--- t ;) Selects the text following NAME up to a delimiting up arrow or a <return> as the current FILENAME for any READ or WRITE operation on CASSETTE, or DISK. The sequence-

```
             0 0 SETNAM <return>
```

will reset the current FILENAME to no name.

8.0 USEFUL UTILITIES

Here is the source for several useful definitions you might need in experimenting with your VIC-20. These are NOT in VICFORTH, but may be entered into an edit screen and saved to CASSETTE or disk.

```
HEX ( --- dl ) ( returns a double number time)

: ?TIME  0 0 0 FFDE SYS >R 100 * + R> ;

DECIMAL  ( xl/yl --- x2/y2 ) ( scales to screen
coord )

:  SCALE 23 - 4 / SWAP 32 - 4 / SWAP;

HEX ( xl/yl --- ) ( move cursor to x,y )
: XY     SWAP 0 0 2SWAP FFF0 SYS  2DROP 2DROP ;

HEX ( --- al )
    ( Returns the base address al of the VIC I/O )
: IOBASE 0 0 0 FFF3 SYS ROT DROP 100 * + ;

    (---) (Load the current edit screen)
: TRY    SCR @ LOAD ;

    (---nl) (Return the amount of free memory)
: FREE   EM @  HERE - ;

    (nl --- ) (Close file number nl)
:CLOSE  0 0 FFC3 SYS 3DROP ;

    ( dl/d2 --- dl/d2/dl )
    ( Duplicate the double number dl over top of )
    ( double number d2. )
:2OVER  >R >R 2DUP R>  R> 2SWAP ;

DECIMAL    (Select the decimal number base.)
    ( --- fl )
    ( Read the joystick and return a boolean flag. )
: J0        37137 C@  4 AND 0= ;
: J1        37137 C@  8 AND 0= ;
: J2        37137 C@ 16 AND 0= ;
: J3        37154 128 TOGGLE  37152 C@ 128 AND 0=
            37154 128 TOGGLE ;
: FIRE       37137 C@ 32 AND 0= ;

    ( --- x/y )
    ( Read the analog paddles, and return x and y. )
: PADDLEXY 36872 @ 256 /MOD ;

    ( --- )
    ( Toggle the auto repeat switch in the system. )
    ( Causes all keys to repeat when held down. )
    ( WARNING: May also cause some keybounce. )
: AUTO.REPEAT 650 128 TOGGLE ;

HEX
    (--- nl)
    ( Read the system status byte, see the )
    ( PROGRAMMERS REFERENCE MANUAL. )
: STATUS 0 0 0 FFB7 SYS 2DROP ;

HEX
    ( Turn on RS-232 port at 300 baud )
    ( Use CLALL to restore OUTPUT to SCREEN )
: SERIAL.ON 6 293 C! 0 294 C! ( Set baud and parity
)
     293 2 SETNAM       ( Tell them to the system )
     2 2 0 SETFLS       ( Set the logical file )
     OPEN               ( Open the channel to RS-232 )
     2 CHKOUT ;         ( Direct output to the channel)

DECIMAL ( Select the Decimal number base. )
        ( Generate a short tone )
: BEEP 10 VOLUME        ( Set the volume of the tone )
```

```
      293 TENOR C!  ( Turn on the TENOR voice =
293)
      1000 0 DO LOOP ( Wait a short time )
      0 TENOR C! ;    ( Turn off the TENOR voice=0 )
```

9.0 VICFORTH ALPHABETIZED GLOSSARY

You can obtain a complete glossary of all fig-FORTH words by requesting it from the FORTH INTEREST GROUP, whose address is given in Section 1.6. Below is a glossary of VICFORTH words that do not appear in the fig glossary.

ŁBLOCKS (n1 ---)
Clears out memory, and sets the number of screen buffers to value n1, after clipping n1 to within the limits (1) and the max memory will allow minus 1.

(MATCH) (addr1/addr2/n1 --- f1)
Compare the strings at addr1 and addr2 for length n1 for equality. Return boolean flag f1 true for match, and false for no-match.

2DROP (d1 ---)
Drop the double number from the top of the stack.

2DUP (d1 --- d1/d1)
Copy the double number on the top of the stack. The duplicate becomes the new top of stack.

2SWAP (d1/d2 --- d2/d1)
The position of the two double numbers on top of the stack is swapped.

3DROP (n1/n2/n3 ---)
Drop the three 16 bit values from the stack, used in system calls to clean off the stack.

<=   (n1/n2 --- f)
Leave a true flag if n1 is less than or equal to n2. Otherwise, a false flag is left.

ALTO   (--- addr1)
Return the address of the alto voice in the VIC chip. Values between 128 and 255 turn this voice on.

ASCII   (--- n1)   P
Smart word, compiles char following into dictionary as a literal if in compile state, else places its value on the stack.

BGROUND   (n1 ---)
Select value n1 as the current BackGROUND color.

BMAX   (--- addr1)   S
A system variable containing the highest value BLK is allowed to assume. If BLK is greater than BMAX, an appropriate error message is issued.

BORDER   (n1 ---)
Select value n1 as the current BORDER color.

CHKOUT   (n1 ---)
Set file number n1 as the current terminal output device for all character output. See PRON, and PRINT. The file number n1 must have been previously opened.

CLALL   (---)
System call to CLear-ALL I/O devices to the default values. See the Programmers Reference Guide.

DL (--- addr1)
A user variable which holds the current READ/WRITE device number, defaults to 1 for cassette.

EDIT (n1 ---)
Select the screen n1 for editing, select the EDITOR vocabulary, and go into the split screen mode. Enables the captured keys for cursor control.

EM (--- addr1)
A system variable, holds the end of memory pointer.

EMIT (b ---)
Transmit byte, b, to the selected output device. OUT is incremented for each character output. The output device is determined by the channel number contained in OUTCH, and what file is opened on that channel.

FL (--- addr1)
A user variable which holds the current file number for all READ/WRITE operations. Defaults to 1 for cassette operation.

FN (--- addr1)
A user variable which holds the name of the current READ/WRITE file. This buffer is 64 characters long.

H. (n1 ---)
Print the value n1 from the stack to the current output device, as an unsigned hexadecimal number.

I/O (--- addr1)

A user variable which returns the beginning address of the I/O vector table area. See 5.2 for the positions of each of the vectored words in the table.

NAME (---)
Accept the text following NAME until a delimiting up-arrow character or a <return> is entered. Place the text into user array FN, with the count byte in the first char of FN.

NOISE (--- addr1)
A system constant which returns the address of the NOISE voice of the VIC chip; values between 128 and 255 turn the voice on.

OPEN (---)
A system command which opens the filename, file number, and device just specified in the last SETFLS and SETNAM commands.

PRINT (---)
Print any output from the line of commands following the print command to the serial bus printer, device £4.

PRON (---)
Send all terminal output to the serial bus printer, device number 4. Can be restored by CLALL.

READ (n1 ---)
Read the first screen from the current device as specified by D£ and F£, into screen n1. Normally will read from cassette, but can be made to read from disk, by changing D£, F£, and specifying a name for the file. Will read named files from cassette,

by specifying a name for the cassette file.

READS (n1/n2 ---)
Read from cassette to screen n1 through n1+n2. The value n2 is the quantity of screens to read. This word can not be made to read screens from disk, since each disk screen must have a different name.

SAVE (addr1/n1 ---)
Save the data or program from address addr1 for a count of n1 bytes to the currently open mass storage device.

SETFLS (n1/n2/n3 ---)
Set the logical file with filenumber n1, device number n2, and secondary command n3 into the operating system.

SETNAM (addr1/n1 ---)
Set the string at address addr1 for count n1 as the name of the current mass storage file name.

SYS (<f1>/n1/n2/n3/addr1 --- <f1>/n4/n5/n6)
Perform a system call to an assembly language routine, at address addr1, with the cpu registers A,X,Y as values n1,n2,n3. Pass optional flag <f1> as the carry state. Return n4,n5,n6 as the cpu registers A,X,Y and return <f1> unmodified.

SLOAD (addr1 ---)
Load the program from the current device, into address addr1. The length of the load is determined by the file being loaded.

SOPRANO (--- addr1)
A system constant, which returns the address of the SOPRANO voice in the VIC chip. Values between 128

and 255 turn this voice on.

TENOR (--- addr1)
A system constant which returns the address of the TENOR voice in the VIC chip. Values between 128 and 255 turn this voice on.

THRU (n1/n2 ---)
Load screens n1 through screen n2, from the screen buffers already in memory. DOES-NOT read any screens from cassette.

U< (n1/n2 --- f1)
Perform an unsigned less than comparison on n1 and n2. Return boolean flag f1 true if n1 is less than n2.

U. (u1 ---)
Print the number u1 to the currently selected output device, as an unsigned number in the current base.

UPORT (--- addr1)
A system constant, returns the address of the USER port.

VECTOR (n1 ---)
A defining word used in the form:

```
      n1 VECTOR nnnn
```

To create a word 'nnnn' which when executed will itself execute the contents of the n1 vectored routine in the I/O table. User created vectored words must start at value 30 decimal and must not be higher than 50.

VOLUME (n1 ---)
Values between 0 and 15 decimal control the VOLUME of all the voices in the VIC chip.

VRESET (---)
Reset all I/O vectors to their default values as held in the default table VWORDS. The values of FL and DL are also reset to 1.

VWORDS (--- addrl)
This word returns the address addrl of the beginning of the initial value vector table in ROM. See section 5.2 for the positions of the vectored words in the table.

WRITE (nl ---)
Write screen nl to the current device, as specified by DL and FL. The written screen will be sent to cassette unless DL and FL are set for disk.

WRITES (nl/n2 ---)
Screens nl through nl+n2 will be written to cassette. Can not be made to write to disk.

fig-FORTH GLOSSARY

This glossary contains all of the word definitions in Release 1 of fig-FORTH. The definitions are presented in the order of their ascii sort.

The first line of each entry shows a symbolic description of the action of the procedure on the parameter stack. The symbols indicate the order in which input parameters have been placed on the stack. Three dashes "---" indicate the execution point; any parameters left on the stack are listed. In this notation, the top of the stack is to the right.

The symbols include:

addr	memory address
b	8 bit byte (i.e. hi 8 bits zero)
c	7 bit ascii character (hi 9 bits zero)
d	32 bit signed double integer, most significant portion with sign on top of stack.
f	boolean flag. 0=false, non-zero=true
ff	boolean false flag=0
n	16 bit signed integer number
u	16 bit unsigned integer
tf	boolean true flag=non-zero

The capital letters on the right show definition characteristics:

C	May only be used within a colon definition. A digit indicates number of memory addresses used, if other than one.
E	Intended for execution only.
L0	Level Zero definition of FORTH-78
L1	Level One definition of FORTH-78
P	Has precedence bit set. Will execute even when compiling.
U	A user variable.

Unless otherwise noted, all references to numbers are for 16 bit signed integers. On 8 bit data bus computers, the high byte of a number is on top of the stack, with the sign in the leftmost bit. For 32 bit signed double numbers, the most significant part (with the sign) is on top.

All arithemetic is implicitly 16 bit signed integer math, with error and under-flow indication unspecified.

!   n addr ---   L0

Store 16 bits of n at address. Pronounced "store".

!CSP

Save the stack position in CSP. Used as part of the compiler security.

d1 --- d2   L0

Generate from a double number d1, the next ascii character which is placed in an output string. Result d2 is the quotient after division by BASE, and is maintained for further processing. Used between <# and #>. See #S.

#>   d --- addr count   L0

Terminates numeric output conversion by dropping d, leaving the text address and character count suitable for TYPE.

#S   d1 --- d2   L0

Generates ascii text in the text output buffer, by the use of #, until a zero double number n2 results. Used between <# and #>.

'   --- addr   P,L0

Used in the form:

    ' nnnn

Leaves the parameter field address of dictionary word nnnn. As a compiler directive, executes in a colon-definition to compile the address as a literal. If the word is not found after a search of CONTEXT and CURRENT, an appropriate error message is given. Pronounced "tick".

(P,L0

Used in the form:

    (cccc)

Ignore a comment that will be delimited by a right parenthesis on the same line. May occur during execution or in a colon-definition. A blank after the leading parenthesis is required.

(.")   C+

The run-time proceedure, compiled by ." which transmits the following in-line text to the selected output device. See ."

(;CODE)   C

The run-time proceedure, compiled by ;CODE, that rewrites the code field of the most recently defined word to point to the following machine code sequence. See ;CODE.

(+LOOP)   n ---   C2

The run-time proceedure compiled by +LOOP, which increments the loop index by n and tests for loop completion. See +LOOP.

(ABORT)

Executes after an error when WARNING is -1. This word normally executes ABORT, but may be altered (with care) to a user's alternative proceedure.

(DO)   C

The run-time proceedure compiled by DO which moves the loop control parameters to the return stack. See DO.

(FIND)   addr1 addr2 --- pfa b tf (ok)
    addr1 addr2 --- ff (bad)

Searches the dictionary starting at the name field address addr2, matching to the text at addr1. Returns parameter field address, length byte of name field and boolean true for a good match. If no match is found, only a boolean false is left.

(LINE)   n1 n2 --- addr count

Convert the line number n1 and the screen n2 to the disc buffer address containing the data. A count of 64 indicates the full line text length.

(LOOP)   C2

The run-time proceedure compiled by LOOP which increments the loop index and tests for loop completion. See LOOP.

(NUMBER)   d1 addr1 --- d2 addr2

Convert the ascii text beginning at addr1+1 with regard to BASE. The new value is accumulated into double number d1, being left as d2. Addr2 is the address of the first unconvertable digit. Used by NUMBER.

*   n1 n2 --- prod   LO
Leave the signed product of two signed numbers.

*/   n1 n2 n3 --- n4   LO
Leave the ratio n4 = n1*n2/n3 where all are signed numbers. Retention of an intermediate 31 bit product permits greater accuracy than would be available with the sequence:
n1 n2 * n3 /

*/MOD   n1 n2 n3 --- n4 n5   LO
Leave the quotient n5 and remainder n4 of the operation n1*n2/n3
A 31 bit intermediate product is used as for */.

+   n1 n2 --- sum   LO
Leave the sum of n1+n2.

+!   n addr ---   LO
Add n to the value at the address. Pronounced "plus-store".

+-   n1 n2 --- n3
Apply the sign of n2 to n1, which is left as n3.

+BUF   addr1 --- addr2 f
Advance the disc buffer address addr1 to the address of the next buffer addr2. Boolean f is false when addr2 is the buffer presently pointed to by variable PREV.

+LOOP   n1 --- (run)
addr n2 --- (compile)   P,C2,LO
Used in a colon-definition in the form:
DO ... n1 +LOOP
At run-time, +LOOP selectively controls branching back to the corresponding DO based on n1, the loop index and the loop limit. The signed increment n1 is added to the index and the total compared to the limit. The branch back to DO occurs until the new index is equal to or greater than the limit (n1>0), or until the new index is equal to or less than the limit (n1<0). Upon exiting the loop, the parameters are discarded and execution continues ahead.

At compile time, +LOOP compiles the run-time word (+LOOP) and the branch offset computed from HERE to the address left on the stack by DO. n2 is used for compile time error checking.

+ORIGIN   n --- addr
Leave the memory address relative by n to the origin parameter area. n is the minimum address unit, either byte or word. This definition is used to access or modify the boot-up parameters at the origin area.

,   n ---   LO
Store n into the next available dictionary memory cell, advancing the dictionary pointer. (comma)

-    n1 n2 --- diff   LO
Leave the difference of n1-n2.

-->   P,LO
Continue interpretation with the next disc screen. (pronounced next-screen).

-DUP n1 -- n1 (if zero)
n1 -- n1 n1 (non-zero) L0

Reproduce n1 only if it is non-zero. This is usually used to copy a value just before IF, to eliminate the need for an ELSE part to drop it.

-FIND --- pfa b tf (found)
--- ff (not found)

Accepts the next text word (delimited by blanks) in the input stream to HERE, and searches the CONTEXT and then CURRENT vocabularies for a matching entry. If found, the dictionary entry's parameter field address, its length byte, and a boolean true is left. Otherwise, only a boolean false is left.

-TRAILING addr n1 --- addr n2

Adjusts the character count n1 of a text string beginning address to suppress the output of trailing blanks. i.e. the characters at addr+n1 to addr+n2 are blanks.

. n --- L0

Print a number from a signed 16 bit two's complement value, converted according to the numeric BASE. A trailing blanks follows. Pronounced "dot".

." P,L0

Used in the form:
." cccc"

Compiles an in-line string cccc (delimited by the trailing ") with an execution proceedure to transmit the text to the selected output device. If executed outside a definition, ." will immediately print the text until the final ". The maximum number of characters may be an installation dependent value. See (.").

.LINE line scr ---

Print on the terminal device, a line of text from the disc by its line and screen number. Trailing blanks are suppressed.

.R n1 n2 ---

Print the number n1 right aligned in a field whose width is n2. No following blank is printed.

/ n1 n2 --- quot L0

Leave the signed quotient of n1/n2.

/MOD n1 n2 --- rem quot L0

Leave the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend.

0 1 2 3 --- n

These small numbers are used so often that is is attractive to define them by name in the dictionary as constants.

0< n --- f L0

Leave a true flag if the number is less than zero (negative), otherwise leave a false flag.

0= n --- f L0

Leave a true flag is the number is equal to zero, otherwise leave a false flag.

0BRANCH f --- C2

The run-time proceedure to conditionally branch. If f is false (zero), the following in-line parameter is added to the interpretive pointer to branch ahead or back. Compiled by IF, UNTIL, and WHILE.

1+ n1 --- n2 L1

Increment n1 by 1.

2+ n1 --- n2

Leave n1 incremented by 2.

: P,E,L0

Used in the form called a colon-definition:

```
        : cccc    ...    ;
```

Creates a dictionary entry defining cccc as equivalent to the following sequence of Forth word definitions '...' until the next ';' or ';CODE'. The compiling process is done by the text interpreter as long as STATE is non-zero. Other details are that the CONTEXT vocabulary is set to the CURRENT vocabulary and that words with the precedence bit set (P) are executed rather than being compiled.

; P,C,L0

Terminate a colon-definition and stop further compilation. Compiles the run-time ;S.

;CODE P,C,L0

Used in the form:

```
    : cccc   ....   ;CODE
          assembly mnemonics
```

Stop compilation and terminate a new defining word cccc by compiling (;CODE). Set the CONTEXT vocabulary to ASSEMBER, assembling to machine code the following mnemonics.

When cccc later executes in the form:

```
      cccc   nnnn
```

the word nnnn will be created with its execution proceedure given by by the machine code following cccc. That is, when nnnn is executed, it does so by jumping to the code after nnnn. An existing defining word must exist in cccc prior to ;CODE.

;S P,L0

Stop interpretation of a screen. ;S is also the run-time word compiled at the end of a colon-definition which returns execution to the calling proceedure.

< n1 n2 --- f L0

Leave a true flag if n1 is less than n2; otherwise leave a false flag.

<# L0

Setup for pictured numeric output formatting using the words:

```
      <#  #  #S  SIGN  #>
```

The conversion is done on a double number producing text at PAD.

<BUILDS C,L0

Used within a colon-definition:

```
    : cccc  <BUILDS  ...
            DOES>    ...   ;
```

Each time cccc is executed, <BUILDS defines a new word with a high-level execution proceedure. Executing cccc in the form:

```
          cccc  nnnn
```

uses <BUILDS to create a dictionary entry for nnnn with a call to the DOES> part for nnnn. When nnnn is later executed, it has the address of its parameter area on the stack and executes the words after DOES> in cccc. <BUILDS and DOES> allow run-time proceedures to written in high-level rather than in assembler code (as required by ;CODE).

= n1 n2 --- f L0

Leave a true flag if n1=n2; otherwise leave a false flag.

> n1 n2 --- f L0

Leave a true flag if n1 is greater than n2; otherwise a false flag.

>R n --- C,L0

Remove a number from the computation stack and place as the most accessable on the return stack. Use should be balanced with R> in the same definition.

? addr -- L0

Print the value contained at the address in free format according to the current base.

?COMP

Issue error message if not compiling.

?CSP

Issue error message if stack position differs from value saved in CSP.

?ERROR f n ---

Issue an error message number n, if the boolean flag is true.

?EXEC

Issue an error message if not executing.

?LOADING

Issue an error message if not loading

?PAIRS n1 n2 ---

Issue an error message if n1 does not equal n2. The message indicates that compiled conditionals do not match.

?STACK

Issue an error message is the stack is out of bounds. This definition may be installation dependent.

?TERMINAL --- f

Perform a test of the terminal keyboard for actuation of the break key. A true flag indicates actuation. This definition is installation dependent.

@ addr --- n L0

Leave the 16 bit contents of address.

ABORT L0

Clear the stacks and enter the execution state. Return control to the operators terminal, printing a message appropriate to the installation.

ABS n --- u L0

Leave the absolute value of n as u.

AGAIN addr n --- (compiling) P,C2,L0

Used in a colon-definion in the form:

BEGIN ... AGAIN

At run-time, AGAIN forces execution to return to corresponding BEGIN. There is no effect on the stack. Execution cannot leave this loop (unless R> DROP is executed one level below).

At compile time, AGAIN compiles BRANCH with an offset from HERE to addr. n is used for compile-time error checking.

ALLOT n --- L0

Add the signed number to the dictionary pointer DP. May be used to reserve dictionary space or re-origin memory. n is with regard to computer address type (byte or word).

AND   n1 n2 --- n2   LO

Leave the bitwise logical and of n1 and n2 as n3.

B/BUF   --- n

This constant leaves the number of bytes per disc buffer, the byte count read from disc by BLOCK.

B/SCR   --- n

This constant leaves the number of blocks per editing screen. By convention, an editing screen is 1024 bytes organized as 16 lines of 64 characters each.

BACK   addr ---

Calculate the backward branch offset from HERE to addr and compile into the next available dictionary memory address.

BASE   --- addr   U,LO

A user variable contaning the current number base used for input and output conversion.

BEGIN   --- addr n (compiling)   P,LO

Occurs in a colon-definition in form:

```
    BEGIN ... UNTIL
    BEGIN ... AGAIN
    BEGIN ... WHILE ... REPEAT
```

At run-time, BEGIN marks the start of a sequence that may be repetitively executed. It serves as a return point from the correspoinding UNTIL, AGAIN or REPEAT. When executing UNTIL, a return to BEGIN will occur if the top of the stack is false; for AGAIN and REPEAT a return to BEGIN always occurs.

At compile time BEGIN leaves its return address and n for compiler error checking.

BL   --- c

A constant that leaves the ascii value for "blank".

BLANKS   addr count ---

Fill an area of memory begining at addr with blanks.

BLK   --- addr   U,LO

A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer.

BLOCK   n --- addr   LO

Leave the memory address of the block buffer containing block n. If the block is not already in memory, it is transferred from disc to which ever buffer was least recently written. If the block occupying that buffer has been marked as updated, it is rewritten to disc before block n is read into the buffer. See also BUFFER, R/W UPDATE FLUSH

BLOCK-READ
BLOCK-WRITE   These are the preferred names for the installation dependent code to read and write one block to the disc.

BRANCH   C2,LO

The run-time proceedure to unconditionally branch. An in-line offset is added to the interpretive pointer IP to branch ahead or back. BRANCH is compiled by ELSE, AGAIN, REPEAT.

BUFFER   n --- addr

Obtain the next memory buffer, assigning it to block n. If the contents of the buffer is marked as updated, it is written to the disc The block is not read from the disc.

The address left is the first cell within the buffer for data storage.

C! b addr ---
Store 8 bits at address. On word addressing computers, further specification is necessary regarding byte addressing.

C, b ---
Store 8 bits of b into the next available dictionary byte, advancing the dictionary pointer. This is only available on byte addressing computers, and should be used with caution on byte addressing minicomputers.

C@ addr --- b
Leave the 8 bit contents of memory address. On word addressing computers, further specification is needed regarding byte addressing.

CFA pfa --- cfa
Convert the parameter field address of a definition to its code field address.

CMOVE from to count ---
Move the specified quantity of bytes beginning at address from to address to. The contents of address from is moved first proceeding toward high memory. Further specification is necessary on word addressing computers.

COLD

The cold start procedure to adjust the dictionary pointer to the minimum standard and restart via ABORT. May be called from the terminal to remove application programs and restart.

COMPILE C2

When the word containing COMPILE executes, the execution address of the word following COMPILE is copied (compiled) into the dictionary. This allows specific compilation situations to be handled in additon to simply compling an execution address (which the interpreter already does).

CONSTANT n --- LO
A defining word used in the form:
n CONSTANT cccc
to create word cccc, with its parameter field containing n. When cccc is later executed, it will push the value of n to the stack.

CONTEXT --- addr U,LO
A user variable containing a pointer to the vocabulary within which dictionary searches will first begin.

COUNT addr1 --- addr2 n LO
Leave the byte address addr2 and byte count n of a message text beginning at address addr1. It is presumed that the first byte at addr1 contains the text byte count and the actual text starts with the second byte. Typically COUNT is followed by TYPE.

CR LO
Transmit a carriage return and line feed to the selected output device.

CREATE

A defining word used in the form:
CREATE cccc

by such words as CODE and CONSTANT to create a dictionary header for a Forth definition. The code field contains the address of the words parameter field. The new word is created in the CURRENT vocabulary.

CSP   ---- addr   U

A user variable temporarily storing the stack pointer position, for compilation error checking.

D+   d1 d2 --- dsum

Leave the double number sum of two double numbers.

D+-   d1 n --- d2

Apply the sign of n to the double number d1, leaving it as d2.

D.   d ---   L1

Print a signed double number from a 32 bit two's complement value. The high-order 16 bits are most accessable on the stack. Conversion is performed according to the current BASE. A blank follows. Pronounced D-dot.

D.R   d n ---

Print a signed double number d right aligned in a field n characters wide.

DABS   d --- ud

Leave the absolute value ud of a double number.

DECIMAL   L0

Set the numeric conversion BASE for decimal input-output.

DEFINITIONS   L1

Used in the form:

cccc DEFINITIONS

Set the CURRENT vocabulary to the CONTEXT vocabulary. In the example, executing vocabulary name cccc made it the CONTEXT vocabulary and executing DEFINITIONS made both specify vocabulary cccc.

DIGIT   c n1 --- n2 tf (ok)
c n1 --- ff   (bad)

Converts the ascii character c (using base n1) to its binary equivalent n2, accompanied by a true flag. If the conversion is invalid, leaves only a false flag.

DLIST

List the names of the dictionary entries in the CONTEXT vocabulary.

DLITERAL   d --- d   (executing)
d ---   (compiling)   P

If compiling, compile a stack double number into a literal. Later execution of the definition containing the literal will push it to the stack. If executing, the number will remain on the stack.

DMINUS   d1 --- d2

Convert d1 to its double number two's complement.

DO   n1 n2 ---   (execute)
addr n ---   (compile) P,C2,L0

Occurs in a colon-definition in form:

DO ... LOOP
DO ... +LOOP

At run time, DO begins a sequence with repetitive execution controlled by a loop limit n1 and an index with initial value n2. DO removes these from the stack. Upon reaching LOOP the index is incremented by one. Until the new index equals or exceeds the limit, execution loops back to just after DO; otherwise the loop parameters are discarded and execution continues ahead. Both n1 and n2 are determined at run-time and may be the result of other operations. Within a loop 'I' will copy the

urrent value of the index to the stack. See I, LOOP, +LOOP, LEAVE.

When compiling within the colon-definition, DO compiles (DO), leaves the following address addr and n for later error checking.

DOES> LO

A word which defines the run-time action within a high-level defining word. DOES> alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following DOES>. Used in combination with <BUILDS. When the DOES> part executes it begins with the address of the first parameter of the new word on the stack. This allows interpretation using this area or its contents. Typical uses include the Forth assembler, multi-diminsional arrays, and compiler generation.

DP ---- addr U,L

A user variable, the dictionary pointer, which contains the address of the next free memory above the dictionary. The value may be read by HERE and altered by ALLOT.

DPL ---- addr U,LO

A user variable containing the number of digits to the right of the decimal on double integer input. It may also be used hold output column location of a decimal point, in user generated formating. The default value on single number input is -1.

DR0
DR1

Installation dependent commands to select disc drives, by preseting OFFSET. The contents of OFFSET is added to the block number in BLOCK to allow for this selection. Offset is supressed for error text so that is may always originate from drive 0.

DROP n --- LO

Drop the number from the stack.

DUMP addr n --- LO

Print the contents of n memory locations beginning at addr. Both addresses and contents are shown in the current numeric base.

DUP n --- n n LO

Duplicate the value on the stack.

ELSE addr1 n1 --- addr2 n2 (compiling) P,C2,LO

Occurs within a colon-definition in the form:

IF ... ELSE ... ENDIF

At run-time, ELSE executes after the true part following IF. ELSE forces execution to skip over the following false part and resumes execution after the ENDIF. It has no stack effect.

At compile-time ELSE emplaces BRANCH reserving a branch offset, leaves the address addr2 and n2 for error testing. ELSE also resolves the pending forward branch from IF by calculating the offset from addr1 to HERE and storing at addr1.

EMIT c --- L0

Transmit ascii character c to the selected output device. OUT is incremented for each character output.

EMPTY-BUFFERS L0

Mark all block-buffers as empty, not necessarily effecting the contents. Updated blocks are not written to the disc. This is also an initialization proceedure before first use of the disc.

ENCLOSE addr1 c --- addr1 n1 n2 n3

The text scanning primitive used by WORD. From the text address addr1 and an ascii delimiting character c, is determined the byte offset to the first non-delimiter character n1, the offset to the first delimiter after the text n2, and the offset to the first character not included. This proceedure will not process past an ascii 'null', treating it as an unconditional delimiter.

END P,C2,L0

This is an 'alias' or duplicate definition for UNTIL.

ENDIF addr n --- (compile) P,C0,L0

Occurs in a colon-definition in form:

```
    IF ... ENDIF
    IF ... ELSE ... ENDIF
```

At run-time, ENDIF serves only as the destination of a forward branch from IF or ELSE. It marks the conclusion of the conditional structure. THEN is another name for ENDIF. Both names are supported in fig-FORTH. See also IF and ELSE.

At compile-time, ENDIF computes the forward branch offset from addr to HERE and stores it at addr. n is used for error tests.

ERASE addr n ---

Clear a region of memory to zero from addr over n addresses.

ERROR line --- in blk

Execute error notification and restart of system. WARNING is first examined. If 1, the text of line n, relative to screen 4 of drive 0 is printed. This line number may be positive or negative, and beyond just screen 4. If WARNING=0, n is just printed as a message number (non disc installation). If WARNING is -1, the definition (ABORT) is executed, which executes the system ABORT. The user may cautiously modify this execution by altering (ABORT). fig-FORTH saves the contents of IN and BLK to assist in determining the location of the error. Final action is execution of QUIT.

EXECUTE addr --

Execute the definition whose code field address is on the stack. The code field address is also called the compilation address.

EXPECT addr count --- L0

Transfer characters from the terminal to address, until a "return" or the count of characters have been received. One or more nulls are added at the end of the text.

FENCE --- addr U

A user variable containing an address below which FORGETting is trapped. To forget below this point the user must alter the contents of FENCE.

FILL addr quan b ---

Fill memory at the address with the specified quantity of bytes b.

FIRST --- n

A constant that leaves the address of the first (lowest) block buffer.

FLD --- addr U

A user variable for control of number output field width. Presently unused in fig-FORTH.

FORGET E,L0

Executed in the form:
FORGET cccc
Deletes definition named cccc from the dictionary with all entries physically following it. In fig-FORTH, an error message will occur if the CURRENT and CONTEXT vocabularies are not currently the same.

FORTH P,L1

The name of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary. Until additional user vocabularies are defined, new user definitions become a part of FORTH. FORTH is immediate, so it will execute during the creation of a colon-definition, to select this vocabulary at compile time.

HERE --- addr L0

Leave the address of the next available dictionary location.

HEX L0

Set the numeric conversion base to sixteen (hexadecimal).

HLD --- addr L0

A user variable that holds the address of the latest character of text during numeric output conversion.

HOLD c --- L0

Used between <# and #> to insert an ascii character into a pictured numeric output string.
e.g. 2E HOLD will place a decimal point.

I --- n C,L0

Used within a DO-LOOP to copy the loop index to the stack. Other use is implementation dependant. See R.

ID. addr ---

Print a definition's name from its name field address.

IF f --- (run-time)
--- addr n (compile) P,C2,L0

Occurs is a colon-definition in form:
IF (tp) ... ENDIF
IF (tp) ... ELSE (fp) ... ENDIF
At run-time, IF selects execution based on a boolean flag. If f is true (non-zero), execution continues ahead thru the true part. If f is false (zero), execution skips till just after ELSE to execute the false part. After either part, execution resumes after ENDIF. ELSE and its false part are optional.; if missing, false execution skips to just after ENDIF.

At compile-time IF compiles 0BRANCH and reserves space for an offset at addr. addr and n are used later for resolution of the offset and error testing.

IMMEDIATE

Mark the most resently made definition so that when encountered at compile time, it will be executed rather than being compiled. i.e. the precedence bit in its header is set. This method allows definitions to handle unusual compiling situations, rather than build them into the fundamental compiler. The user may force compilation of an immediate definition by preceeding it with [COMPILE].

IN --- addr LO

A user variable containing the byte offset within the current input text buffer (terminal or disc) from which the next text will be accepted. WORD uses and moves the value of IN.

INDEX from to ---

Print the first line of each screen over the range from, to. This is used to view the comment lines of an area of text on disc screens.

INTERPRET

The outer text interpreter which sequentially executes or compiles text from the input stream (terminal or disc) depending on STATE. If the word name cannot be found after a search of CONTEXT and then CURRENT it is converted to a number according to the current base. That also failing, an error message echoing the name with a " ?" will be given.
Text input will be taken according to the convention for WORD. If a decimal point is found as part of a number, a double number value will be left. The decimal point has no other purpose than to force this action.
See NUMBER.

KEY --- c LO

Leave the ascii value of the next terminal key struck.

LATEST --- addr

Leave the name field address of the topmost word in the CURRENT vocabulary.

LEAVE C,LO

Force termination of a DO-LOOP at the next opportunity by setting the loop limit equal to the current value of the index. The index itself remains unchanged, and execution prodeeds normally until LOOP or +LOOP is encountered.

LFA   pfa --- lfa

Convert the parameter field address of a dictionary definition to its link field address.

LIMIT   ---- n

A constant leaving the address just above the highest memory available for a disc buffer. Usually this is the highest system memory.

LIST   n ---   LO

Display the ascii text of screen n on the selected output device. SCR contains the screen number during and after this process.

LIT   --- n   C2,LO

Within a colon-definition, LIT is automatically compiled before each 16 bit literal number encountered in input text. Later execution of LIT causes the contents of the next dictionary address to be pushed to the stack.

LITERAL   n --- (compiling)   P,C2,LO

If compiling, then compile the stack value n as a 16 bit literal. This definition is immediate so that it will execute during a colon definition. The intended use is:

```
   : xxx   [ calculate ]  LITERAL  ;
```

Compilation is suspended for the compile time calculation of a value. Compilation is resumed and LITERAL compiles this value.

LOAD   n ---   LO

Begin interpretation of screen n. Loading will terminate at the end of the screen or at ;S. See ;S and -->.

LOOP   addr n --- (compiling) P,C2,LO

Occurs in a colon-definition in form:

```
    DO  ...  LOOP
```

At run-time, LOOP selectively controls branching back to the corresponding DO based on the loop index and limit. The loop index is incremented by one and compared to the limit. The branch back to DO occurs until the index equals or exceeds the limit; at that time, the parameters are discarded and execution continues ahead.

At compile-time, LOOP compiles (LOOP) and uses addr to calculate an offset to DO. n is used for error testing.

M*   n1 n2 --- d

A mixed magnitude math operation which leaves the double number signed product of two signed number.

M/   d n1 --- n2 n3

A mixed magnitude math operator which leaves the signed remainder n2 and signed quotient n3, from a double number dividend and divisor n1. The remainder takes its sign from the dividend.

M/MOD ud1 u2 --- u3 ud4

An unsigned mixed magnitude math operation which leaves a double quotient ud4 and remainder u3, from a double dividend ud1 and single divisor u2.

MAX n1 n2 --- max LO

Leave the greater of two numbers.

MESSAGE n ---

Print on the selected output device the text of line n relative to screen 4 of drive 0. n may be positive or negative. MESSAGE may be used to print incidental text such as report headers. If WARNING is zero, the message will simply be printed as a number (disc un-available).

MIN n1 n2 --- min LO

Leave the smaller of two numbers.

MINUS n1 --- n2 LO

Leave the two's complement of a number.

MOD n1 n2 --- mod LO

Leave the remainder of n1/n2, with the same sign as n1.

MON

Exit to the system monitor, leaving a re-entry to Forth, if possible.

MOVE addr1 addr2 n ---

Move the contents of n memory cells (16 bit contents) beginning at addr1 into n cells beginning at addr2. The contents of addr1 is moved first. This definition is appropriate on on word addressing computers.

NEXT

This is the inner interpreter that uses the interpretive pointer IP to execute compiled Forth definitions. It is not directly executed but is the return point for all code proceedures. It acts by fetching the address pointed by IP, storing this value in register W. It then jumps to the address pointed to by the address pointed to by W. W points to the code field of a definition which contains the address of the code which executes for that definition. This usage of indirect threaded code is a major contributor to the power, portability, and extensibility of Forth. Locations of IP and W are computer specific.

NFA pfa --- nfa

Convert the parameter field address of a definition to its name field.

NUMBER addr --- d

Convert a character string left at addr with a preceeding count, to a signed double number, using the current numeric base. If a decimal point is encountered in the text, its position will be given in DPL, but no other affect occurs. If numeric conversion is not possible, an error message will be given.

OFFSET --- addr U

A user variable which may contain a block offset to disc drives. The contents of OFFSET is added to the stack number by BLOCK. Messages by MESSAGE are independent of OFFSET. See BLOCK, DR0, DR1, MESSAGE.

OR n1 n2 -- or LO

Leave the bit-wise logical or of two 16 bit values.

OUT --- addr U

A user variable that contains a value incremented by EMIT. The user may alter and examine OUT to control display formating.

OVER n1 n2 --- n1 n2 n1 LO

Copy the second stack value, placing it as the new top.

PAD --- addr LO

Leave the address of the text output buffer, which is a fixed offset above HERE.

PFA nfa --- pfa

Convert the name field address of a compiled definition to its parameter field address.

POP

The code sequence to remove a stack value and return to NEXT. POP is not directly executable, but is a Forth re-entry point after machine code.

PREV ---- addr

A variable containing the address of the disc buffer most recently referenced. The UPDATE command marks this buffer to be later written to disc.

PUSH

This code sequence pushes machine registers to the computation stack and returns to NEXT. It is not directly executable, but is a Forth re-entry point after machine code.

PUT

This code sequence stores machine register contents over the topmost computation stack value and returns to NEXT. It is not directly executable, but is a Forth re-entry point after machine code.

QUERY

Input 80 characters of text (or until a "return") from the operators terminal. Text is positioned at the address contained in TIB with IN set to zero.

QUIT L1

Clear the return stack, stop compilation, and return control to the operators terminal. No message is given.

R --- n

Copy the top of the return stack to the computation stack.

R# --- addr U

A user variable which may contain the location of an editing cursor, or other file related function.

R/W addr blk f ---

The fig-FORTH standard disc read-write linkage. addr specifies the source or destination block buffer, blk is the sequential number of the referenced block; and f is a flag for f=0 write and f=1 read. R/W determines the location on mass storage, performs the read-write and performs any error checking.

R> --- n L0

Remove the top value from the return stack and leave it on the computation stack. See >R and R.

R0 --- addr U

A user variable containing the initial location of the return stack. Pronounced R-zero. See RP!

REPEAT addr n --- (compiling) P,C2

Used within a colon-definition in the form:

BEGIN ... WHILE ... REPEAT

At run-time, REPEAT forces an unconditional branch back to just after the correspoinding BEGIN.

At compile-time, REPEAT compiles BRANCH and the offset from HERE to addr. n is used for error testing.

ROT n1 n2 n3 --- n2 n3 n1 L0

Rotate the top three values on the stack, bringing the third to the top.

RP!

A computer dependent proceedure to initialize the return stack pointer from user variable R0.

S->D n --- d

Sign extend a single number to form a double number.

S0 --- addr U

A user variable that contains the initial value for the stack pointer. Pronounced S-zero. See SP!

SCR --- addr U

A user variable containing the screen number most recently reference by LIST.

SIGN n d --- d LO

Stores an ascii "-" sign just before a converted numeric output string in the text output buffer when n is negative. n is discarded, but double number d is maintained. Must be used between <# and #>.

SMUDGE

Used during word definition to toggle the "smudge bit" in a definitions' name field. This prevents an uncompleted definition from being found during dictionary searches, until compiling is completed without error.

SP!

A computer dependent proceedure to initialize the stack pointer from S0.

SP@ --- addr

A computer dependent proceedure to return the address of the stack position to the top of the stack, as it was before SP@ was executed. (e.g. 1 2 SP@ @ . . . would type 2 2 1)

SPACE LO

Transmit an ascii blank to the output device.

SPACES n --- LO

Transmit n ascii blanks to the output device.

STATE --- addr LO,U

A user variable containg the compilation state. A non-zero value indicates compilation. The value itself may be implementation dependent.

SWAP n1 n2 --- n2 n1 LO

Exchange the top two values on the stack.

TASK

A no-operation word which can mark the boundary between applications. By forgetting TASK and re-compiling, an application can be discarded in its entirety.

THEN P,C0,L0

An alias for ENDIF.

TIB --- addr U

A user variable containing the address of the terminal input buffer.

TOGGLE addr b ---

Complement the contents of addr by the bit pattern b.

TRAVERSE addr1 n --- addr2

Move across the name field of a fig-FORTH variable length name field. addr1 is the address of either the length byte or the last letter. If n=1, the motion is toward hi memory; if n=-1, the motion is toward low memory. The addr2 resulting is address of the other end of the name.

TRIAD   scr ---

Display on the selected output device the three screens which include that numbered scr, begining with a screen evenly divisible by three. Output is suitable for source text records, and includes a reference line at the bottom taken from line 15 of screen4.

TYPE   addr count ---   LO

Transmit count characters from addr to the selected output device.

U*   u1 u2 --- ud

Leave the unsigned double number product of two unsigned numbers.

U/   ud u1 --- u2 u3

Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.

UNTIL   f --- (run-time)
addr n --- (compile)   P,C2,LO

Occurs within a colon-definition in the form:

    BEGIN ... UNTIL

At run-time, UNTIL controls the conditional branch back to the corresponding BEGIN. If f is false, execution returns to just after BEGIN; if true, execution continues ahead.

At compile-time, UNTIL compiles (OBRANCH) and an offset from HERE to addr. n is used for error tests.

UPDATE   LO

Marks the most recently referenced block (pointed to by PREV) as altered. The block will subsequently be transferred automatically to disc should its buffer be required for storage of a different block.

USE   --- addr

A variable containing the address of the block buffer to use next, as the least recently written.

USER   n ---   LO

A defining word used in the form:

    n USER cccc

which creates a user variable cccc. The parameter field of cccc contains n as a fixed offset relative to the user pointer register UP for this user variable. When cccc is later executed, it places tbe sum of its offset and the user area base address on the stack as the storage address of that particular variable.

VARIABLE E,LU

A defining word uaed in the form:

n VARIABLE cccc

When VARIABLE ia executed, it creates the definition cccc with its parameter field initialized to n. When cccc is later executed, the address of its parameter field (containing n) ia left on the atack, so that a fetch or atore may acceas thia location.

VOC-LINK --- addr U

A user variable containing the addreaa of a field in the definition of the most recently created vocabulary. All vocabulary names are linked by these fields to allow control for FORGETting thru multiple vocabularys.

VOCABULARY E,L

A defining word used in the form:

VOCABULARY cccc

to create a vocabulary definition cccc. Subsequent uae of cccc will make it the CONTEXT vocabulary which ia aearched first by INTERPRET. The sequence "cccc DEFINITIONS" will alao make cccc the CURRENT vocabulary into which new definitions are placed.

In fig-FORTH, cccc will be ao chained as to include all definitions of the vocabulary in which cccc ia itself defined. All vocabularya ulitmately chain to Forth. By convention, vocabulary namea are to be declared IMMEDIATE. See VOC-LINK.

VLIST

List the names of the definitions in the context vocabulary. "Break" will terminate the liating.

WARNING --- addr U

A user variable containing a value controlling messages. If = 1 diac is present, and screen 4 of drive 0 ia the baae location for mesaagea. If = 0, no disc ia preaent and messages will be preaented by number. If = -1, execute (ABORT) for a uaer apecified proceedure. See MESSAGE, ERROR.

WHILE f --- (run-time)

ad1 n1 --- ad1 n1 ad2 n2 P,C2

Occurs in a colon-definition in the form:

BEGIN ... WHILE (tp) ... REPEAT

At run-time, WHILE selecta conditional execution based on boolean flag f. If f is true (non-zero), WHILE continuea execution of the true part thru to REPEAT, which then branches back to BEGIN. If f is false (zero), execution akipa to just after REPEAT, exiting the atructure.

At compile time, WHILE emplacea (OBRANCH) and leaves ad2 of the reaerved offset. The stack values will be resolved by REPEAT.

WIDTH --- addr U

In fig-FORTH, a user variable containing the maximum number of lettara aaved in the compilation of a definitions' name. It must be 1 thru 31, with a default value of 31. The name character count and its natural characters are aaved, up to the value in WIDTH. The value may be changed at any time within the above limits.

WORD c --- L0

Read the next text characters from the input stream being interpreted, until a delimiter c ia found, atoring the packed character atring begining at the dictionary buffer HERE. WORD leaves the character count in the first byte, the characters, and enda with two or more blanks. Leading occurances of c are ignored. If BLK is zero, text is taken from the terminal input buffer, otherwise from the disc block stored in BLK. See BLK, IN.

X

This is paeudonym for the "null" or dictionary entry for a name of one character of ascii null. It is the execution proceedure to terminate interpretation of a line of text from the terminal or within a disc buffer, as both buffera alwaya have a null at the end.

XOR n1 n2 --- xor L1

Leave the bitwise logical excluaive-or of two values.

[P,L1

Used in a colon-definition in form:

```
    : xxx   [  words  ]   more  ;
```

Suspend compilatioo. The worda after [are executed, not compiled. This allows calculatioo or compilation exceptioos before reauming compilation with]. See LITERAL,].

[COMPILE] P,C

Used in a colon-defioition in form:

```
   : xxx   [COMPILE]  FORTH  ;
```

[COMPILE] will force the compilation of ao immediate defininitioo, that would otherwiae execute during compilation. The above example will aelect the FORTH vocabulary wheo xxx executes, rather than at compile time.

] L1

Resume compilatioo, to the completion of a colon-definition. See [.

COPYRIGHT NOTICE

Although we make every attempt to verify the accuray of this document, we cannot assume any liability for errors or omissions. No warranty or other guarantee can be given as to the accuracy or suitability of this software for a particular purpose, nor can we be liable for any loss or damage arising from the use of the same.

HES

Human Engineered Software
71 Park Lane
Brisbane, California 94005
Telephone 415-468-4110

HES

VIC FORTH

by Tom Zimmer

C 301